华中科技大学

操作系统原理课程实验报告

姓　　名：　　李嘉鹏

学　　院：　　计算机科学与技术学院

专　　业：　　数据科学与大数据技术

班　　级：　　大数据 2101 班

学　　号：　　U202115652

指导教师：　　周正勇

分数	
教师签名	

2023 年 12 月 6 日

目　录

实验一 相对路径（挑战实验 4-1）

1.1 实验目的

在 PKE 操作系统内核中完善对文件相对路径的支撑，实现通过相对路径对文件的访问。具体要求为：

①通过修改 PKE 文件系统代码，提供解析相对路径的支持。具体来说，用户应用程序在使用任何需要传递路径字符串的函数时，都可以传递相对路径而不会影响程序的功能。这一功能可以大大方便用户对文件的操作，因为不必记住绝对路径。

②完成用户层 pwd 函数和 cd 函数。其中，pwd 函数（即 present working directory）需要读取进程当前的工作目录，cd 函数（即 change pwd）需要实现切换进程当前工作目录的功能。

1.2 实验内容

A.实验分析

首先需要理解相对路径的形式与实现原理。相对路径是形如“./file”、“./dir/file”以及“../dir/file”的路径形式。与绝对路径总是从根目录开始逐级指定文件或目录在目录树中的位置不同，相对路径从进程的当前工作目录开始，对一个文件或目录在目录树中的位置进行描述。

从路径字符串形式的角度来看，相对路径的起始处总是两种特殊目录之一：“.”或“..”。其中“.”代指进程的当前工作目录，“..”代指进程当前工作目录的父目录。例如“./file”的含义为位于进程当前工作目录下的 file 文件；“../dir/file”的含义为当前进程工作目录父目录下的 dir 目录下的 file 文件。

从文件系统的角度来说，相对路径的实现首先依赖于每个进程中所维护的当前工作目录。其次，具体的文件系统需要对相对路径访问方式提供支持。

本实验中共有 35 个可修改的操作系统内核代码文件，如图 1 所示；在实际过程中，我对 proc_file.c、proc_file.h、syscall.c、syscall.h、vfs.c 这 5 个代码文件进行了修改，添加了 SYS_user_rcwd（用户层 pwd）和 SYS_user_ccwd（用户层 cd）两个函数的实现逻辑和细节。

图 1

B.代码具体实现

（1）SYS_user_rcwd

需要实现用户层读取当前工作目录的功能，可以通过三个步骤来实现：

①通过 user_va_to_pa 函数将用户态的虚拟地址 path 转换为用户态的物理地址 pa；

②从当前进程的文件表 current->pfiles 中获取当前进程工作目录的路径 current->pfiles->cwd->name；

③使用 memcpy 函数将当前进程的工作目录路径复制到用户态地址空间中。

完整代码如下所示。

```
ssize_t sys_user_rcwd(uint64 path) {
  uint64 pa = (uint64)user_va_to_pa((pagetable_t)(current->pagetable), (void*)path);
  memcpy((char*)pa, current->pfiles->cwd->name, strlen(current->pfiles->cwd->name));
  return 0;
}
```

（2）SYS_user_ccwd

需要更改当前进程的工作目录路径，可以通过四个步骤来实现：

①将用户态的虚拟地址 path 转换为用户态的物理地址 pa；

②将当前进程工作目录的路径 current->pfiles->cwd->name 赋值给 temp，便于进行路径的操作；

③根据用户提供的路径，分情况进行处理：

a.若路径以 .. 开头，表示返回上一级目录，则进行相应操作；

b.若路径以 / 开头，说明为绝对路径，则直接将其赋值给当前工作目录路径；

c.对于其它情况，则进行路径的拼接操作，并更新当前工作目录路径；

④更新当前进程的打开文件状态。

在具体实现上，通过逐字符读取用户提供的路径 data 可以实现上面③的分类操作。若 data 的前两个字符为“..”，说明需要返回父目录，此时从 temp 最后开始向前寻找上一级目录，找到后将其后的路径屏蔽掉。此后还需要注意进行目录的拼接，将用户提供的 path 中的路径接在上一级目录后。

如果 data 的开头字符为“/”，则代表了绝对路径，直接将其替换工作目录路径即可。最后，在得到目标路径后，还需要更新当前进程工作目录的路径，并调用 reset_opened_files_state()函数更新当前进程的文件状态。完整代码如下所示。

```
ssize_t sys_user_ccwd(uint64 path) {
  // 将用户态的虚拟地址转换为用户态的物理地址
  uint64 pa = (uint64)user_va_to_pa((pagetable_t)(current->pagetable), (void*)path);
  char* data = (char*)pa;
  // 处理 cd
  char temp[MAX_PATH_LEN];
  memset(temp, '\0', MAX_PATH_LEN);
  memcpy(temp, current->pfiles->cwd->name, strlen(current->pfiles->cwd->name));
  if (data[0] == '.') {
    if (data[1] == '.') {
      // cd ..
      int len = strlen(temp);
      for (int i = len - 1; i >= 0; i--) {
        if (temp[i] == '/') {
          temp[i] = '\0';
          break;
        }
        temp[i] = '\0';
      }
      len = strlen(temp);
      if (len == 0) {
        temp[0] = '/';
```

```
        temp[1] = '\0';
      }
    }

    int offset = 0;
    if (strlen(temp) == 1)
      offset = 1;
    int begin = 0;
    for (int i = 0; i < strlen(data); i++)
      if (data[i] == '/') {
        begin = i;
        break;
      }
    if (begin != 0) {
      int len = strlen(temp);
      for (int i = begin + offset; i < strlen(data); i++)
        temp[len++] = data[i];
    }

    memset(current->pfiles->cwd->name, '\0', MAX_PATH_LEN);
    memcpy(current->pfiles->cwd->name, temp, strlen(temp));
  }
  else if (data[0] == '/')
    // cd /xxx
    memcpy(current->pfiles->cwd->name, data, strlen(data));
  reset_opened_files_state();
  return 0;
}
```

reset_opened_files_state()函数负责清空当前进程的文件状态。首先将当前进程的已打开文件数量设置为 0，表示当前没有打开的文件。然后需要遍历当前进程的所有文件描述符 file descriptor，将所有打开文件的状态设置为 FD_NONE，表示文件描述符当前没有被使用。函数代码如下所示。

```
void reset_opened_files_state() {
  current->pfiles->nfiles = 0;
  for (int fd = 0; fd < MAX_FILES; ++fd)
    current->pfiles->opened_files[fd].status = FD_NONE;
}
```

1.3 实验调试及心得

A.实验结果

如图 2 所示，执行 obj_relativepath 和 obj_relativepath2 时均能返回正确的结果，即可以正确实现相对路径下的文件读 read、写 write、切换目录 cwd 操作。

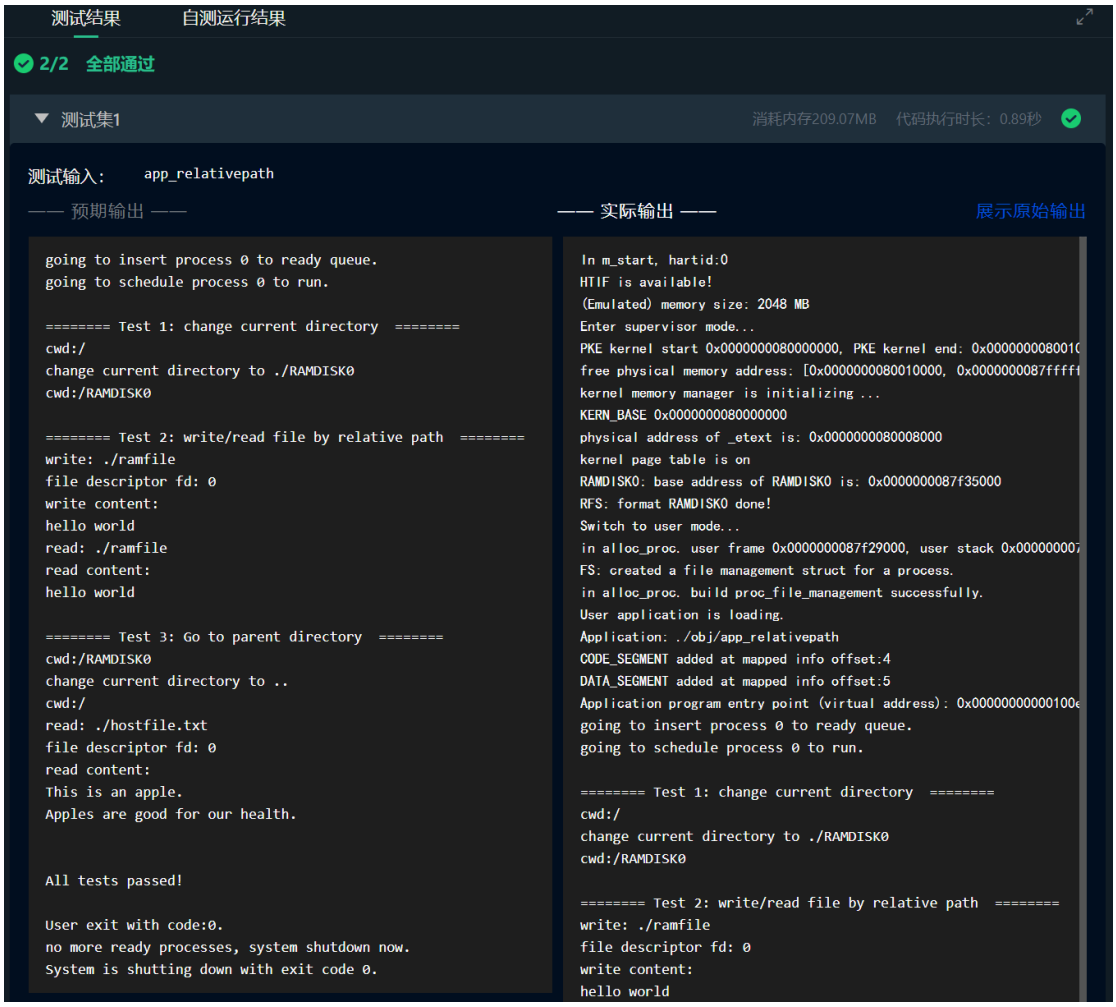

图 2

B.调试与心得

第四个挑战实验相对来说思路比较清晰，因为实验要求中已经指明了需要完成用户层 pwd 函数和 cd 函数，因此只需要针对性地实现这两个功能即可。当然具体写起来还是遇到了一些问题，例如一开始在将物理地址 pa 赋值给 data 时，由于指针的语法错了，导致一直报错“不能创建新的数组”；还有在进行目录路径的拼接时，一开始遗漏了“./”的格式，导致拼接出来的路径变成了“.//xxx”，导致访问失败。不过通过不断构造测试用例并修改函数，这些问题最终都解决了。

总体来说，这学期的操作系统实验让我收获很大，很多课上的理论知识实际上并不好用几行代码实现，在真实情况中要考虑大量的临界条件、判断可能发生的异常，这也让我更加深刻地认识到了底层操作系统的综合性与复杂性。